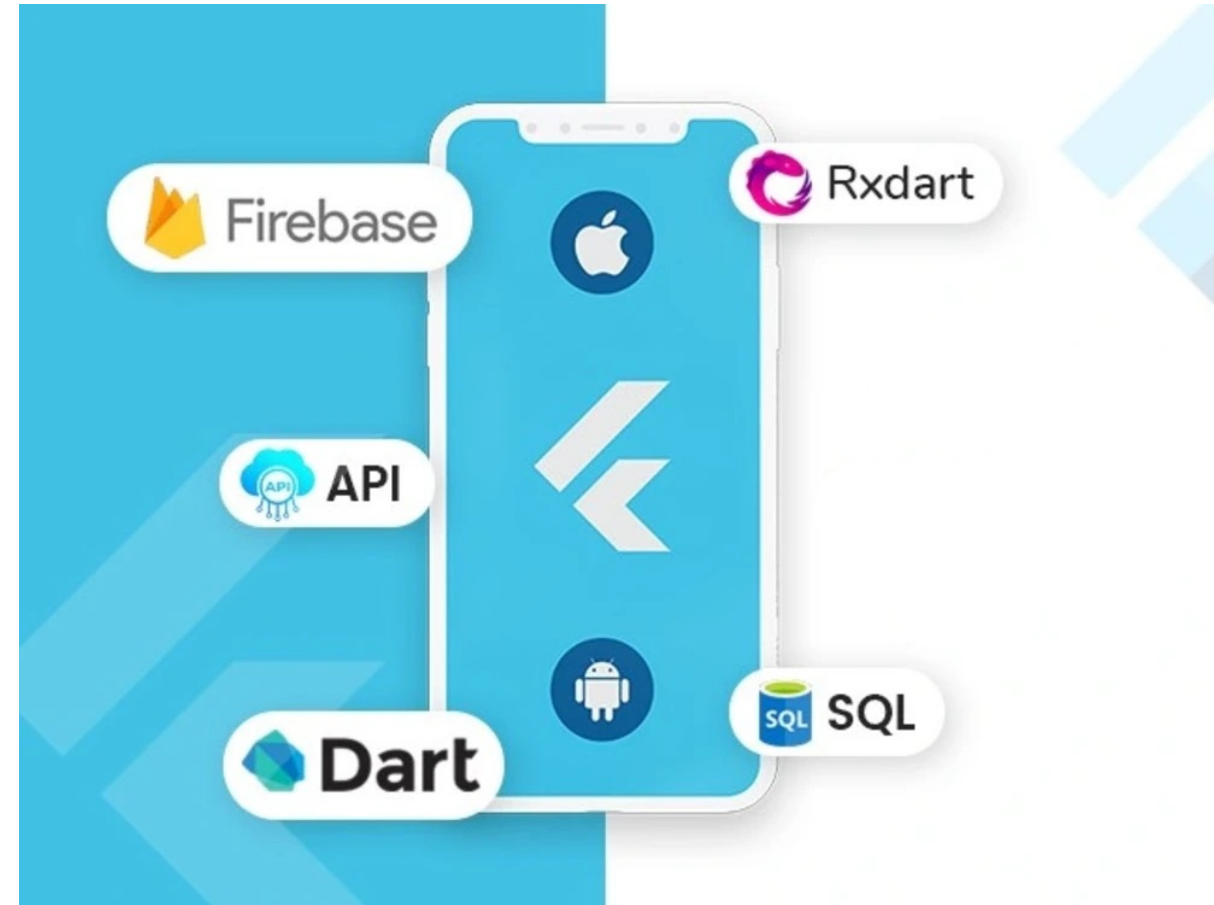


## BÀI 6

# LẬP TRÌNH MẠNG & CƠ SỞ DỮ LIỆU

**GV: ThS. BÙI PHÚ KHUYÊN**



**\* Cập nhật: 24.09.2024**

- ☐ Giới thiệu JSON và xử lý JSON
- ☐ Làm việc với REST APIs
- ☐ Shared Preferences
- ☐ SQLite và SQFLite
- ☐ Firebase



- **JSON** (JavaScript Object Notation) là một định dạng dữ liệu nhẹ và dễ đọc, được sử dụng rộng rãi để trao đổi dữ liệu giữa máy khách (**client**) và máy chủ (**server**).
- **JSON** sử dụng các cặp **key - value** để lưu trữ dữ liệu và hỗ trợ các cấu trúc dữ liệu như đối tượng (**object**) và mảng (**array**).

key phải nằm trong  
dấu nháy kép " "

{

"name": "Bùi Phú Khuyến",

"job": "Giảng viên",

"age": 26

}

value có thể chứa  
chuỗi/số/mảng/đối  
tượng...

- **Object**: Được biểu diễn bằng dấu ngoặc nhọn {}. Các cặp **key-value** nằm bên trong dấu ngoặc nhọn.

{

**"firstName"**: **"Khuyen"**,**"lastName"**: **"Bui"**,**"age"**: **26**

}

- **Array:** Được biểu diễn bằng dấu ngoặc vuông [ ]. Một mảng chứa nhiều giá trị hoặc đối tượng.

```
[  
  {  
    "name": "Khuyên",  
    "age": 26,  
    "gender": "Nam"  
  },  
  {  
    "name": "Cường",  
    "age": 30,  
    "gender": "Nam"  
  },  
]
```

- **Nested Object:** Là đối tượng lồng bên trong đối tượng khác. .

```
{  
  "khuyen": {  
    "name": "Khuyên",  
    "age": 26,  
    "gender": "Nam"  
  },  
  "cuong": {  
    "name": "Cường",  
    "age": 30,  
    "gender": "Nam"  
  },  
}
```

- **Serialize** là quá trình chuyển đổi một đối tượng (Object) Dart thành một chuỗi JSON.
- Quá trình này rất hữu ích khi ta cần gửi dữ liệu từ ứng dụng Flutter đến máy chủ

```
class User {  
  String name;  
  int age;  
  
  User({required this.name, required this.age});  
  
  // Phương thức chuyển đổi từ Object Dart sang JSON  
  Map<String, dynamic> toJson() => {  
    'name': name,  
    'age': age,  
  };  
}
```

**Bước 1: Tạo Model (Object)**

```
import 'dart:convert';  
  
void main() {  
  User user = User(name: 'Khuyên', age: 26);  
  String jsonString = jsonEncode(user.toJson());  
  print(jsonString);  
  // Output: {"name":"Khuyên","age":26}  
}
```

**Bước 2: Chuyển đổi Object sang Json**

- **Deserialize** là quá trình chuyển đổi một chuỗi JSON thành một đối tượng Dart.
- Thường dùng khi ta nhận dữ liệu từ máy chủ và cần sử dụng nó trong ứng dụng Flutter.

```
class User {  
  String name;  
  int age;  
  
  User({required this.name, required this.age});  
  
  // Phương thức để chuyển đổi từ JSON sang đối tượng Dart  
  factory User.fromJson(Map<String, dynamic> json) {  
    return User(  
      name: json['name'],  
      age: json['age'],  
    );  
  }  
}
```

**Bước 1: Tạo Model (Object)**

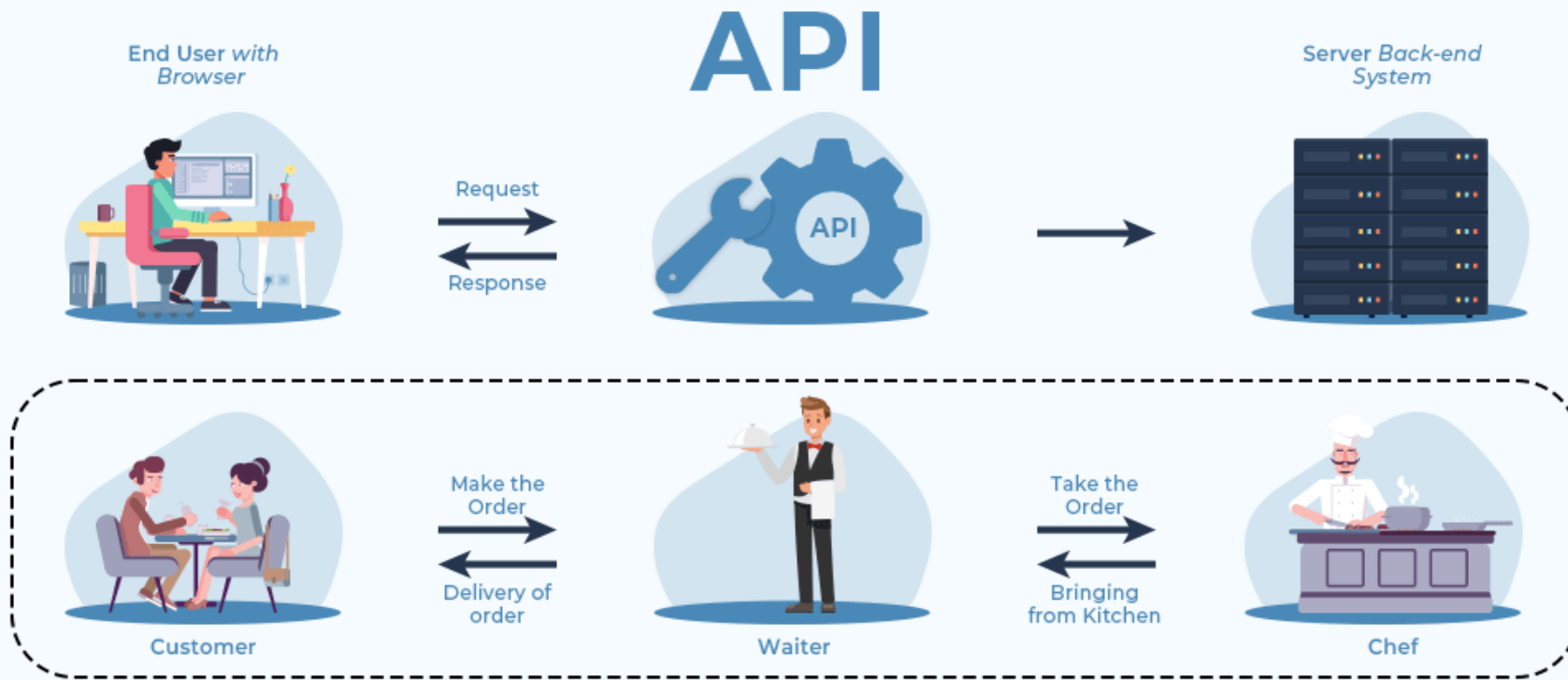
```
import 'dart:convert';
```

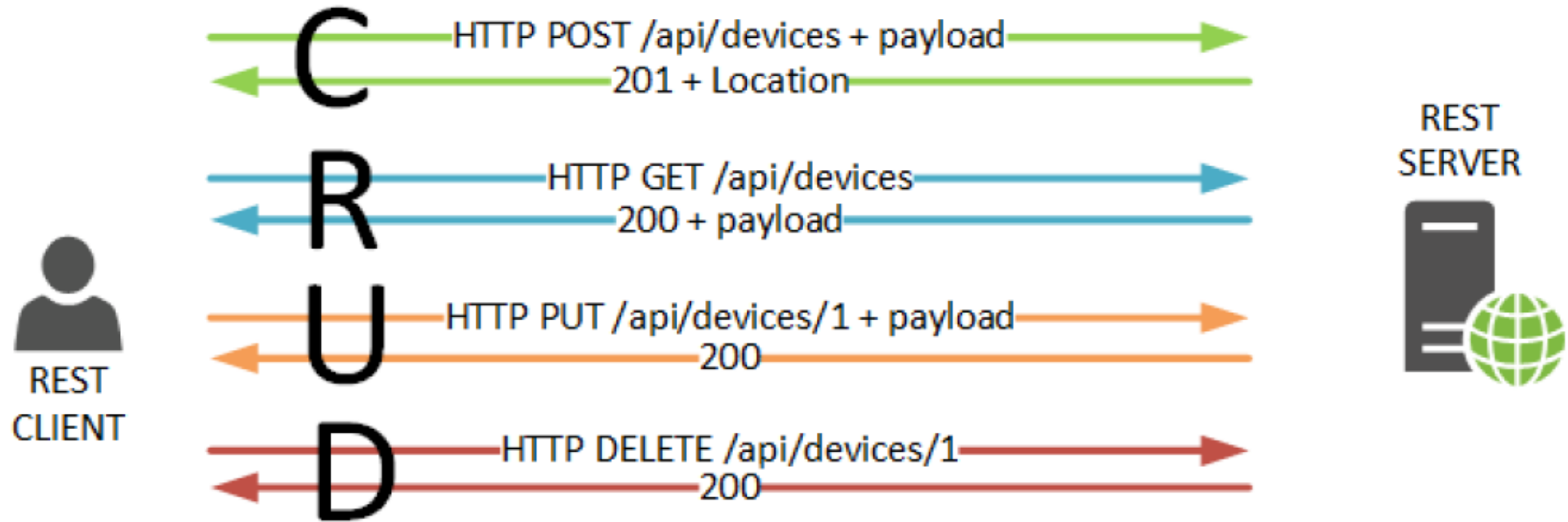
```
void main() {  
  String jsonString = '{"name":"Khuyên","age":26}';  
  Map<String, dynamic> userMap = jsonDecode(jsonString);  
  User user = User.fromJson(userMap);  
  print('Name: ${user.name}, Age: ${user.age}');  
  //Output: Name: Khuyên, Age: 26  
}
```

**Bước 2: Chuyển đổi Json sang Object**



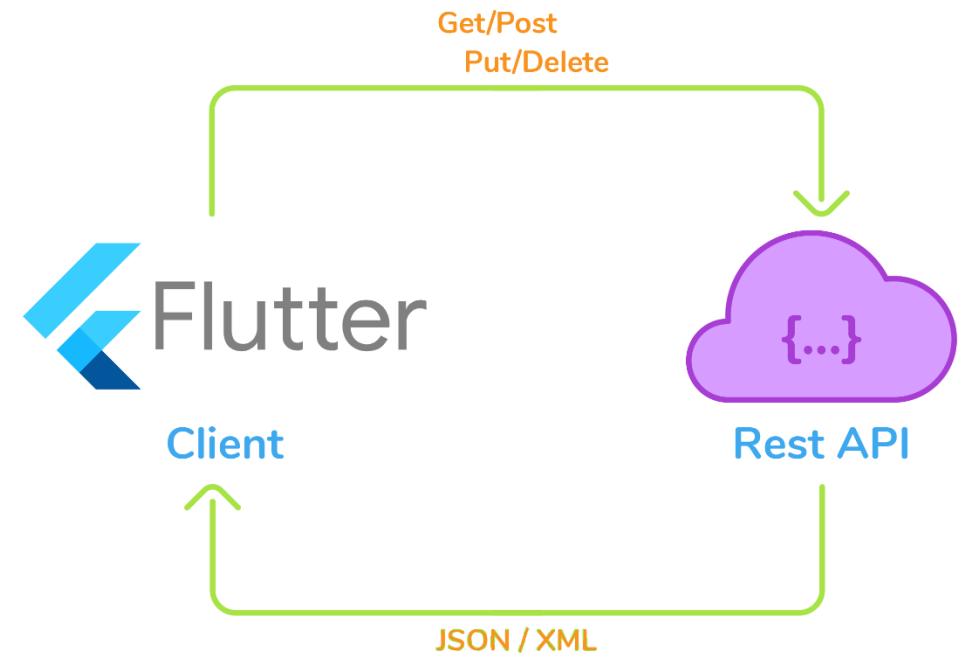
**REST APIs** là một loại **API** sử dụng các phương thức **HTTP** (như **GET**, **POST**, **PUT**, **DELETE**) để thực hiện các thao tác **CRUD** (Create, Read, Update, Delete) trên tài nguyên





- Bước 1: Thiết lập cách gọi API từ Flutter: Cần sử dụng thư viện http/dio...
- Bước 2: Tạo yêu cầu HTTP (Get, Post, Put...) trong Flutter
- Bước 3: Xử lý dữ liệu trả về từ API
- Bước 4: Hiển thị dữ liệu API ra giao diện Flutter
- Bước 5: Xử lý lỗi khi gọi API

Lưu ý: Ngoài các bước trên có thể bổ sung các bước khác (nếu cần) như: Token và Xác thực; COR...



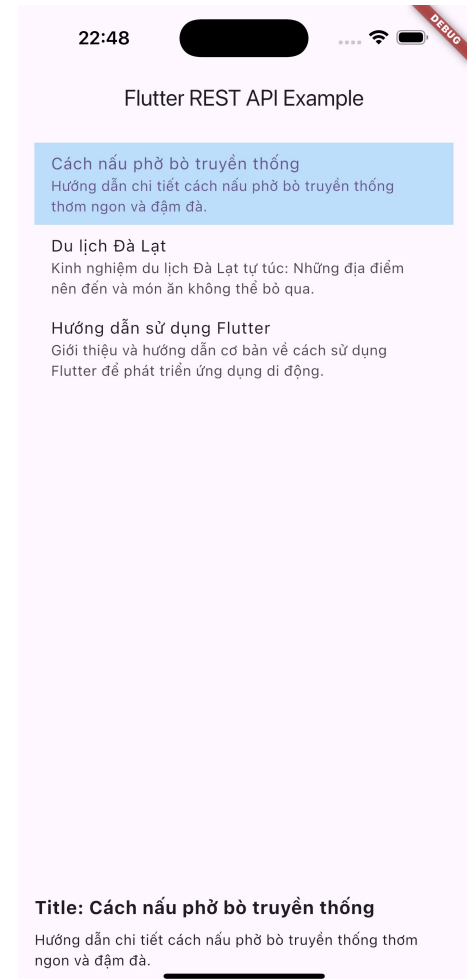
## BÀI TẬP VẬN DỤNG

### Nội dung kiến thức

- Sử dụng thư viện **http** để tương tác với **API**
- **Future** và **FutureBuilder** trong **Flutter**
- Làm việc với **API REST (GET)**
- Xử lý **JSON** và hiển thị dữ liệu

```
1 {  
2   {  
3     "id": 1,  
4     "title": "Cách nấu phở bò truyền thống",  
5     "body": "Hướng dẫn chi tiết cách nấu phở bò truyền thống thơm ngon và đậm đà."  
6   },  
7  
8   {  
9     "id": 2,  
10    "title": "Du lịch Đà Lạt",  
11    "body": "Kinh nghiệm du lịch Đà Lạt tự túc: Những địa điểm nên đến và món ăn không thể bỏ qua."  
12  },  
13  
14  {  
15    "id": 3,  
16    "title": "Hướng dẫn sử dụng Flutter",  
17    "body": "Giới thiệu và hướng dẫn cơ bản về cách sử dụng Flutter để phát triển ứng dụng di động."  
18  }  
19 }
```

[my-json-server.typicode.com/buiphukhuyen/api/posts/](https://my-json-server.typicode.com/buiphukhuyen/api/posts/)



## THIẾT LẬP THƯ VIỆN ĐỂ GỌI API (http)

http 1.2.2

Published 2 months ago • dart.dev Dart 3 compatible

SDK DART FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

7.8K

Readme Changelog Example Installing Versions Scores

Use this package as a library

Depend on it

Run this command:

With Dart:

```
$ dart pub add http
```

With Flutter:

```
$ flutter pub add http
```

This will add a line like this to your package's pubspec.yaml (and run an implicit `dart pub get`):

## TẠO MODEL POST (BÀI VIẾT)

```
[
  {
    "id": 1,
    "title": "Cách nấu phở bò truyền thống",
    "body": "Hướng dẫn chi tiết cách nấu phở bò truyền thống thơm ngon và đậm đà."
  },
  {
    "id": 2,
    "title": "Du lịch Đà Lạt",
    "body": "Kinh nghiệm du lịch Đà Lạt tự túc: Những địa điểm nên đến và món ăn không thể bỏ qua."
  },
  {
    "id": 3,
    "title": "Hướng dẫn sử dụng Flutter",
    "body": "Giới thiệu và hướng dẫn cơ bản về cách sử dụng Flutter để phát triển ứng dụng di động."
  }
]
```

<https://dart-quicktype.netlify.app/>

```
lib > model > post.dart > ...
1  class Post {
2    Post({
3      required this.id,
4      required this.title,
5      required this.body,
6    });
7
8    final int? id;
9    final String? title;
10   final String? body;
11
12   //Hàm chuyển đổi từ Json sang đối tượng Post
13   factory Post.fromJson(Map<String, dynamic> json) {
14     return Post(
15       id: json["id"],
16       title: json["title"],
17       body: json["body"],
18     );
19   }
20
21   //Hàm chuyển đổi đối tượng Post thành Json
22   Map<String, dynamic> toJson() => {
23     "id": id,
24     "title": title,
25     "body": body,
26   };
27 }
```

## TẠO APISERVICE ĐỂ KẾT NỐI VỚI REST API

lib > model > api\_service.dart > ...

```
1 import 'package:http/http.dart' as http;
2 import 'dart:convert';
3 import 'post.dart';
4
5 class ApiService {
6   final String baseUrl = 'https://my-json-server.typicode.com/buiphukhuyen/api/posts';
7
8   // Lấy toàn bộ bài viết
9   Future<List<Post>> fetchAllPosts() async {
10     final response = await http.get(Uri.parse(baseUrl));
11
12     if (response.statusCode == 200) {
13       List<dynamic> body = jsonDecode(response.body);
14       return body.map((json) => Post.fromJson(json)).toList();
15     } else {
16       throw Exception('Có lỗi khi tải toàn bộ bài viết');
17     }
18   }
19 }
```

```
19
20 // Lấy một bài viết cụ thể theo ID
21 Future<Post> fetchPost(int id) async {
22   final response = await http.get(Uri.parse('$baseUrl/$id'));
23
24   if (response.statusCode == 200) {
25     return Post.fromJson(jsonDecode(response.body));
26   } else {
27     throw Exception('Có lỗi khi tải chi tiết bài viết');
28   }
29 }
30 }
```

<https://my-json-server.typicode.com/buiphukhuyen/api/posts>

## HIỂN THỊ DANH SÁCH BÀI VIẾT VỚI FUTUREBUILDER/LISTVIEW.BUILDER {1}

```
import 'package:flutter/material.dart';  
import 'package:flutter_restapi/model/api_service.dart';  
import 'package:flutter_restapi/model/post.dart';
```

```
class PostScreen extends StatefulWidget {  
  const PostScreen({super.key});  
  
  @override  
  _PostScreenState createState() => _PostScreenState();  
}
```

```
class _PostScreenState extends State<PostScreen> {  
  late Future<List<Post>>  
  | futurePosts; // Biến lưu trữ Future để lấy danh sách bài viết từ API
```

```
|  
| @override  
| void initState() {  
|   super.initState();  
|   futurePosts =  
|   | ApiService().fetchAllPosts(); // Khởi tạo Future để lấy toàn bộ bài viết  
| }
```



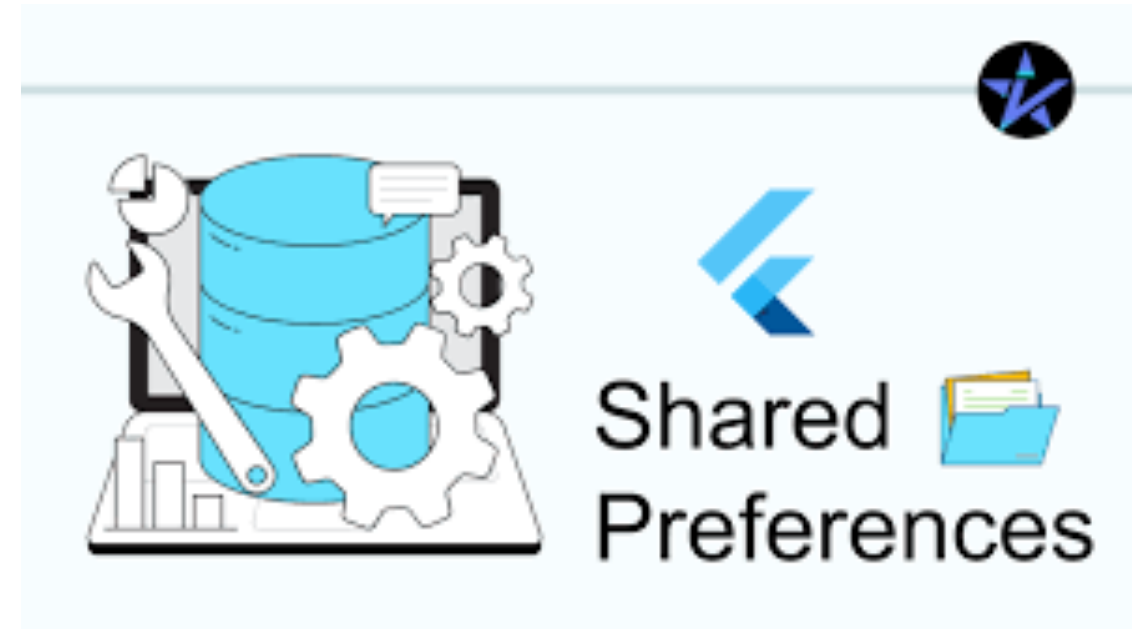
## HIỂN THỊ DANH SÁCH BÀI VIẾT VỚI FUTUREBUILDER/LISTVIEW.BUILDER {2}

```
Expanded(  
  child: FutureBuilder<List<Post>>(  
    future: futurePosts, // Future để lấy dữ liệu từ API  
    builder: (context, snapshot) {  
      // Nếu đang trong trạng thái chờ (dữ liệu chưa về)  
      if (snapshot.connectionState == ConnectionState.waiting) {  
        return const Center(  
          child:  
            CircularProgressIndicator(); // Hiển thị vòng tròn chờ  
        );  
      }  
      // Nếu có lỗi trong quá trình lấy dữ liệu  
      else if (snapshot.hasError) {  
        return Text('Lỗi: ${snapshot.error}'); // Hiển thị lỗi  
      }  
      // Nếu dữ liệu đã được tải thành công  
      else if (snapshot.hasData) {  
        List<Post> posts = snapshot.data!; // Lấy danh sách bài viết
```

```
        List<Post> posts = snapshot.data!; // Lấy danh sách bài viết  
        return ListView.builder(  
          itemCount: posts.length, // Số lượng bài viết  
          itemBuilder: (context, index) {  
            return ListTile(  
              title: Text(posts[index]  
                .title!), // Hiển thị tiêu đề của bài viết // Text  
              subtitle: Text(posts[index]  
                .body!), // Hiển thị nội dung của bài viết // Text  
            ); // ListTile  
          },  
        ); // ListView.builder  
      }  
      // Trường hợp không có dữ liệu (danh sách rỗng)  
      else {  
        return const Text('Không có dữ liệu');  
      }  
    },  
  ), // FutureBuilder  
) , // Expanded
```

Tiếp theo: Sinh viên tiếp tục hoàn thiện chức năng lấy chi tiết 1 bài viết

- **Shared Preferences** là một plugin trong Flutter dùng để **lưu trữ dữ liệu đơn giản** dưới dạng **key-value** trên các nền tảng khác nhau
- Là phương pháp lưu trữ dữ liệu **không đồng bộ**, giúp lưu trữ các dữ liệu nhỏ như cấu hình ứng dụng, trạng thái đăng nhập của người dùng
- **Không** được khuyến khích sử dụng để lưu trữ các **dữ liệu quan trọng** do cơ chế **ghi dữ liệu vào bộ nhớ máy**. Nếu gỡ ứng dụng sẽ mất toàn bộ dữ liệu.



## THIẾT LẬP THƯ VIỆN VÀO DỰ ÁN

### shared\_preferences 2.3.2

Published 38 days ago • flutter.dev Dart 3 compatible

SDK FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

9.5K

Readme Changelog Example Installing Versions Scores

### Use this package as a library

Depend on it

Run this command:

With Flutter:

```
$ flutter pub add shared_preferences
```

## CÁCH LƯU TRỮ DỮ LIỆU

Sử dụng các phương thức **set**

```
void saveData() async {  
    //Khởi tạo Shared Preferences  
    final prefs = await SharedPreferences.getInstance();  
    //Lưu trữ giá trị int 10 vào key 'counter'  
    await prefs.setInt('counter', 10);  
    //Lưu trữ giá trị bool true vào key 'repeat'  
    await prefs.setBool('repeat', true);  
    //Lưu trữ giá trị double 1.5 vào key 'decimal'  
    await prefs.setDouble('decimal', 1.5);  
    //Lưu trữ giá trị String Start vào key 'action'  
    await prefs.setString('action', 'Start');  
    //Lưu trữ danh sách String vào key 'items'  
    await prefs.setStringList('items', <String>['Earth', 'Moon', 'Sun']);  
}
```

## CÁCH LẤY/XOÁ DỮ LIỆU

Sử dụng các phương thức **get/remove**

```
void loadData() async {  
    final prefs = await SharedPreferences.getInstance();  
    //Shared Preferences sẽ tìm kiếm và lấy data với key truyền vào. Nếu không tồn tại giá trị chứa  
    key đó sẽ trả về null  
    int? counter = prefs.getInt('counter');  
    bool? repeat = prefs.getBool('repeat');  
    double? decimal = prefs.getDouble('decimal');  
    String? action = prefs.getString('action');  
    List<String>? items = prefs.getStringList('items');  
}  
  
void removeData() async {  
    final prefs = await SharedPreferences.getInstance();  
    await prefs.remove('counter'); // Xóa dữ liệu với key 'counter'  
}
```

12:51

Đăng nhập

Email  
khuyenpb@gmail.com


Mật khẩu  
.....

Đăng nhập

Chưa đăng nhập => LoginScreen()

12:52

Hồ sơ



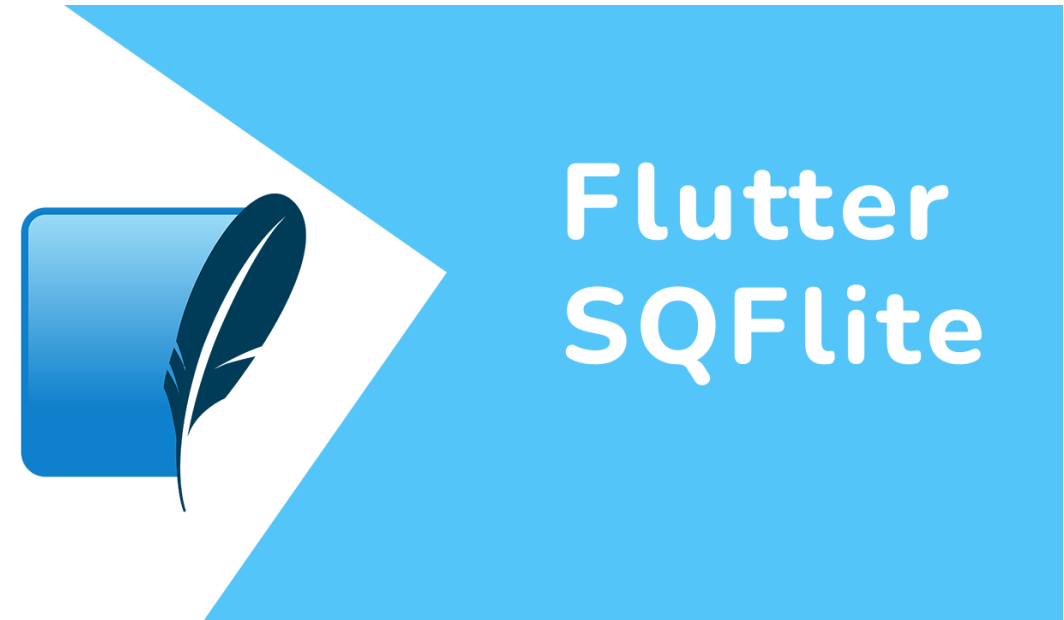
**Tên người dùng**

Email: email@example.com  
Số điện thoại: 0123456789

Trang chủ    Hồ sơ

Đã đăng nhập => DashboardScreen()

- **SQLite** là một hệ quản trị **cơ sở dữ liệu quan hệ** nhúng, nhẹ, dễ sử dụng và không yêu cầu một server riêng biệt.
- **SQLite** thường được sử dụng trong ứng dụng di động để **lưu trữ dữ liệu cục bộ** như thông tin người dùng, cấu hình ứng dụng hoặc các dữ liệu khác mà ta muốn lưu trữ một cách có tổ chức.
- Thư viện **sqflite** là một plugin Flutter phổ biến cho phép tương tác với **SQLite** một cách dễ dàng. **sqflite** cung cấp các phương thức để thực hiện các thao tác cơ bản như thêm, sửa, xóa và truy vấn dữ liệu



## THIẾT LẬP THƯ VIỆN VÀO DỰ ÁN

### sqlite 2.3.3+1

Published 4 months ago • [tekartik.com](https://pub.dev/packages/sqlite) Dart 3 compatible

[SDK](#) [FLUTTER](#) [PLATFORM](#) [ANDROID](#) [IOS](#) [MACOS](#)

 4.9K

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

### Use this package as a library

Depend on it

Run this command:

With Flutter:

```
$ flutter pub add sqlite
```



## MỞ/ĐÓNG CƠ SỞ DỮ LIỆU

- SQLite database được lưu trữ dưới dạng tệp trong hệ thống tệp, được xác định bởi một đường dẫn.

```
import 'package:sqflite/sqflite.dart';  
  
// Mở cơ sở dữ liệu  
  
var db = await openDatabase('demo.db');  
  
  
// Đóng cơ sở dữ liệu  
  
await db.close();
```

## THAO TÁC VỚI CSDL (CRUD) {1}

```
await db.execute('''  
    CREATE TABLE Test (  
        id INTEGER PRIMARY KEY,  
        name TEXT,  
        value INTEGER,  
        num REAL  
    )  
''');
```

Tạo bảng

```
await db.insert('Test',  
{ 'name': 'John', 'value':  
123, 'num': 1.23});
```

Thêm dữ liệu cho bảng

## THAO TÁC VỚI CSDL (CRUD) {2}

```
await db.update('Test',  
{ 'name': 'Jane', 'value': 999 },  
where: 'id = ?',  
whereArgs: [1]);
```

Cập nhật dữ liệu

```
await db.delete('Test',  
where: 'id = ?',  
whereArgs: [1]);
```

Xoá dữ liệu

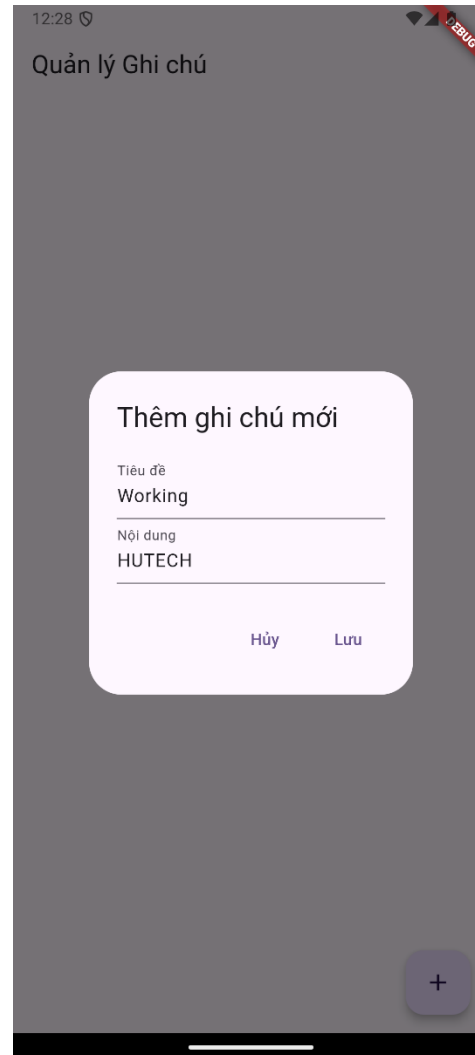
## THAO TÁC VỚI CSDL (CRUD) {3}

```
var result = await db.query('Test');  
print(result);
```

Truy vấn toàn bộ

```
var result = await db.query(  
  'Test',  
  where: 'id = ?',  
  whereArgs: [1]);
```

Truy vấn có điều kiện



# Cảm ơn

## Các bạn đã chú ý lắng nghe

---





# Q&A